Forecasting the production from Danish solar panels with Long Short-Term Memory Neural Networks

In collaboration with Energinet

Martin Kamp Dalgaard(1), Tobias Kallehauge(2), and Jonas Villadsen(3)
Mathematical Engineering, Aalborg University, December 18, 2018

Abstract— **The national transmission system operator is responsible for making sure that the power available in the power grid matches the demand as closely as possible, which involves forecasting the amount of power being produced. This paper investigates how neural networks can be used to forecast the power production from solar panels. It is made in collaboration with the Danish transmission system operator, Energinet, who has supplied the paper with measurements of the power production and weather forecasts in Denmark for 2015, 2016, and 2017. Training data are used to train different architectures of Long Short-Term Memory neural networks, which are then tested on the last 5 days of every month in 2017. The LSTM architectures included in this paper are among others a shallow and stacked LSTM, whose performances are compared to an autoregressive model with exogenous input currently employed by Energinet. The improvements over the autoregressive model are 53.21% and 55.85% for the shallow and stacked LSTM, respectively.**

I. Introduction

Energinet is the Danish transmission system operator (TSO) and facilitates the energy grid and energy market in Denmark as a part of a larger collaboration between countries – in particular the Nordic and Baltic countries on the Nord Pool Spot market which is the largest in Europe. A part of Energinet's role as TSO is making sure that the national energy demand is matched with the potential in the power grid at any time. If there either is a surplus or a shortage of power Energinet regulates the power being produced through e.g. power plants, wind turbines or by buying or selling power at energy markets [**demarticle**]. A key part of being able to regulate the grid in advance requires forecasting both the power production and consumption accurately. This involves predicting the power produced by solar panels, which can be particularly difficult due to the weather. Energinet is interested in making 1 hour forecasts due to time aspects of the energy market, which in this paper is referred to as the forecast horizon h. Furthermore, Energinet considers Denmark divided into a western and eastern region for which the production is forecasted collectively.

This paper focuses on forecasting the solar power production (SPP). The data supplied by Energinet are:

1) Master data, which consists of the solar panels' total installed capacity for each municipality and the municipalities' positions. The installed capacity describes how much power the solar panels are able to produce in [kW], and these are updated daily since solar panels are frequently being installed and removed.
2) The total SPP in each municipality in a 15 minute resolution from January 1, 2015, to December 31, 2017. It should be noted that only data from a few selected solar panels in each municipality are available, and these data are then extrapolated by Energinet to represent the total production. As these data are the best data Energinet has access to, they are considered to be the target data to be modelled. These data are denoted by P and measured in [MWh/15min].
3) Weather forecasts with prognoses of the *global horizontal irridiance* (GHI), which expresses how much energy reaches the Earth per square meter ($[W/m^2]$), and prognoses of temperature ($[K]$), wind speed ($[m/s]$), and wind direction in degrees from north. These data are available in a grid with 15×15 km spacing covering all of Denmark, in a 1 hour resolution with a 56 hour horizon, and are updated 4 times a day. These data are linearly interpolated to a 15 minute resolution in order to make the data easier to handle.

Energinet currently employs a form of *autoregressive model with exogenous input* (ARX) to forecast the SPP, denoted by P. The exogenous input is the GHI, and analyses show that it is reasonable to assume that the GHI and the SPP are linearly correlated [**LinReg**]. The exogenous part of the model at time t and forecast horizon h can be stated as $P_{\mathrm{X}}(h,t) = \alpha(t+h) \cdot \mathrm{GHI}(t+h)$ where α is a weight found through the ordinary least squares and $\mathrm{GHI}(t+h)$ is the forecasted GHI at time $t+h$. In this paper the weight α depends on the municipality whose SPP is forecasted, the time of day, and the season. The autoregressive part of the model is the relation $P_{\mathrm{AR}}(h,t) = P_{\mathrm{AR}}(0,t) \cdot w(h,t)$ with an autoregressive weight w that depends on the forecast horizon h and time of day t. Thus the ARX model is

$$P_{\mathrm{ARX}}(h,t) = P_{\mathrm{X}}(h,t) + P_{\mathrm{AR}}(h,t). \qquad (1)$$

The ARX model gives Energinet good performance but with the popularity of more advanced machine learning techniques, it is of interest to search for a model with an even better performance. Examples of advanced machine learning methods in the literature are [**prev˙sol2**], where spatial correlation is included using a *general linear model*

(1) mkda15@student.aau.dk
(2) tkalle15@student.aau.dk
(3) jvill14@student.aau.dk

(GML) of SPP in different locations, and [**prev˙sol1**], where a *feed-forward neural network* (FNN) and a simple *recurrent neural network* (RNN) are used. The paper [**prev˙sol1**] found that the preferred type of neural network architecture depends on the setup of the training data. The reason for the wide use of RNNs for time series prediction is the fact that the recurrent loops in these allow them to share the same weights across different time steps and hence consider correlation with previous information. However, the networks are trained by minimizing the gradients, which for RNNs can result in vanishing gradients. This is avoided with two advanced RNN structures, *long short-term memory* (LSTM) and *gated recurrent unit* (GRU) neural networks, which uses certain gates to control the flow of information [**Goodfellow**]. In the paper [**prev˙sol3**] these structures are used to predict the SPP, and the performance is compared with an autoregressive model, an FNN, and a *support vector machine* (SVM). In the paper [**prev˙sol3**] the best performances in training time and root mean square error is found for GRU and LSTM with no significant difference between the two. This paper will experiment with LSTM networks and in particular how added depth influences the performances of these networks.

In section II the basics behind LSTM networks and different LSTM architectures are described. Hyperparameters for these networks are tuned in sections III-A and III-B followed by a performance analysis in section III-C. The methods and results from this paper are discussed in section IV, and the paper is finally concluded in section V.

The developed software can be found in the public Github repository https://github.com/JonasVilladsen/P7_libraries.

II. Methods

RNN is a general term for neural networks that handles time series of various lengths by sharing hidden states across time. Simple RNNs are single layer perceptron models with an autoregressive term, and in the training of these the gradient is found by the *back-propagation through time* (BPTT) algorithm where the gradient of the loss function with respect to the weights is computed. However, the derivative at time t with respect to time 0 tends to decrease exponentially as t increases, which means that a simple RNN is usually incapable of learning long-term dependencies [**Bengio**] [**Goodfellow**]. LSTM neural networks have memory gates that control the flow of information through time which – in combination with a cell state – can avoid the vanishing gradient problem. An LSTM memory cell is shown in figure 1.

The LSTM cell is characterized by the set of equations

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right) \tag{2}$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right) \tag{3}$$

$$\tilde{\mathbf{c}}_t = \phi\left(\mathbf{W}_{x\tilde{c}}\mathbf{x}_t + \mathbf{W}_{h\tilde{c}}\mathbf{h}_{t-1} + \mathbf{b}_{\tilde{c}}\right) \tag{4}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{5}$$

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o\right) \tag{6}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \phi(\mathbf{c}_t), \tag{7}$$

where $\mathbf{i}_t$, $\mathbf{o}_t$, and $\mathbf{f}_t$ are the input, output, and forget gates at time t, respectively, $\tilde{\mathbf{c}}_t$ and $\mathbf{c}_t$ are the candidate cell state and

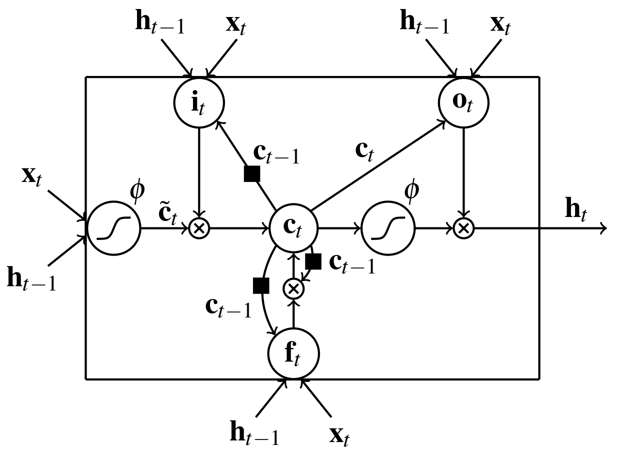

Fig. 1: An LSTM memory cell at time t [**graves˙LSTM**].

cell state, respectively, σ is the "recurrent activation", ϕ is the activation function, and $\mathbf{W}$ are different weight matrices. Usually, σ is the (hard) Sigmoid and ϕ is the identity for regression. The Sigmoid function maps to $[0,1]$, and thus the interpretation of the gates is reducing the flow of information such that 0 is none and 1 is all.

A. Input & model

Firstly, the input to the LSTM network is considered. The paper [**Norm**] shows that the convergence rate is increased by normalizing or whitening the input data. In order to accommodate this the residual signal is calculated through the SPP P and the exogenous part P_{X} from (1):

$$P_{\mathrm{res}} = P - P_{\mathrm{X}}. \tag{8}$$

Since P_{X} is fitted with the unbiased least squares estimator, the resulting residual signal will be zero mean although the variance changes throughout the day. Thus the input will be a vector of P_{res} and P_{X} for all Danish municipalities. The structure of the LSTM network should allow for it to remember relevant information from previous time steps through the cell state but historical input can also be included at each time step to potentially enable this to a higher degree. In the LSTM network a number of historical inputs n (including the input at time t) is allowed, and n is referred to as the number of autoregressive terms. The input vector at time t and forecast horizon h is then

$$\mathbf{x}_{t,h} = \begin{bmatrix}\mathbf{P}^T_{\mathrm{res},t} & \cdots & \mathbf{P}^T_{\mathrm{res},t-(n-1)} & \mathbf{P}^T_{\mathrm{X},t+h} & \cdots & \mathbf{P}^T_{\mathrm{X},t+h-(n-1)}\end{bmatrix}^T$$

where $\mathbf{P}^T_{\mathrm{res},i}$ and $\mathbf{P}^T_{\mathrm{X},i}$ are vectors with data for all the Danish municipalities.

An RNN such as an LSTM is limited in its ability to model nonlinear systems due to only having the recurrent activation function as nonlinear part but adding depth to an RNN is not as straightforward as for other types of neural networks. Different approaches are considered in the following.

One simple method to add depth is to replace the model for P_{X} with an FNN with dense layers. This model is referred to as a *deep GHI LSTM* (dLSTM) and is illustrated to the far left in figure 2. Here $\mathbf{x}$ is split into the two data types 'GHI' and 'SPP'. 'D' illustrates an FNN of dense layers, and 'h/c' is the hidden/cell state in the LSTM cell. The recurrent part of the LSTM cell is illustrated with the square, which delays $\mathbf{h}_t$ and $\mathbf{c}_t$ by one sample. The possible advantage of this is a more advanced model for P_X but also that the training algorithm can see the entire system which could increase the performance. Other possibilities are the *stacked LSTM* (sLSTM), *deep output* (DO), *deep state* (DS), and *deep transition shortcut* (DTs). These architectures are also shown in figure 2, which except for the dLSTM are inspired by [**Stallion3**].

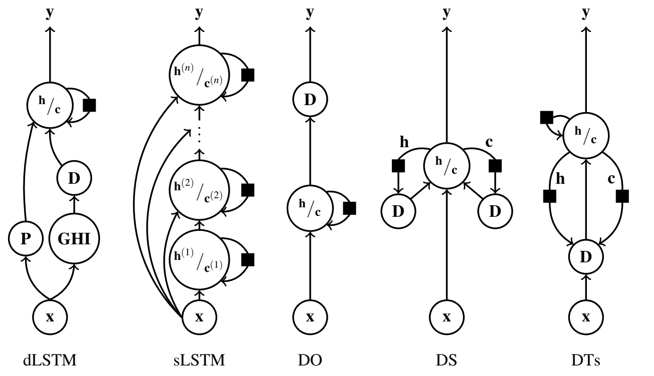

Fig. 2: The 5 different LSTM architectures used in this paper.

The DS and DTs architectures can also be combined into a general *deep transition deep state* (DTDS) architecture, where both the input $\mathbf{x}$ and the outputs from the LSTM cell are fed into dense layers.

The networks mentioned here all have advantages and disadvantages when compared to each other. The DS and DTs networks allow for a deep transition between two consecutive hidden states, and DTs further has the ability to do some feature extraction because the input is being fed straight into a deep FNN together with the previous states. The DO architecture simply allows the output to be deep. The sLSTM is more connected in time due to the hidden states being both connected to the previous hidden state in time and the previous hidden state in depth; however, each transition between consecutive states can be considered shallow as argued by [**Stallion3**]. Whether a deep RNN will have benefits over simpler structures can be hard to determine since RNNs are already indirectly deep in time. However, [**Stallion3**] found that both the DTs and DO performed better than the shallow and stacked RNN when dealing with language and music modelling, and the performance analysis in section III-C will examine if that is also the case for short-time forecasts of the SPP.

B. Training

An optimization algorithm is used to fit the weights in a neural network with a loss function for evaluation. For RNNs the choice of loss function is problem-specific, and in case of regression the *mean squared error* (MSE) or *mean absolute error* (MAE) can be used. This will be explored further in section III.

A commonly used optimization algorithm is the *Root mean square propagation* (RMS-prop) [**Goodfellow**]. RMS-prop is a subclass of the gradient descent algorithm which incorporates an adaptive learning rate η with a moving average of the gradient $\mathbf{g}_t = \nabla_{\mathbf{W}}\mathscr{L}(\hat{\mathbf{y}}_t(\mathbf{W}), \mathbf{y}_t)$ of the loss function with respect to the weight matrix $\mathbf{W}$. For the RMS-prop algorithm, an element-wise moving average M of $\mathbf{g}_t^2$ is computed as $M[\mathbf{g}_t^2] = \gamma M[\mathbf{g}_{t-1}^2] + (1-\gamma)\mathbf{g}_t^2$, where $0 < \gamma < 1$ is a momentum parameter, and the network's weights can then be updated as [**Goodfellow**]:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \frac{\eta}{\sqrt{M[\mathbf{g}_t^2] + \varepsilon}} \odot \mathbf{g}_t. \qquad (9)$$

The gradient for RNNs – and LSTMs in particular – can at any given time be calculated with the BPTT algorithm which uses a recursive approach to efficiently compute the gradient.

When fitting a model using RMS-prop it is usually necessary to go over the training data multiple times in order for the loss function to converge to a minimum. One *epoch* refers to fitting the training data one time, and in each epoch the training data will be divided into batches (groups of 10 days in this paper) to avoid memory issues and potentially allow for parallelisation of the algorithm.

Because the networks in this paper should account for seasonal fluctuations it is not necessarily desired to just train the networks on all of the training data. Instead, data from 2015 and 2016 are used exclusively for training data, and 2017 will be split into training/validation data as well as testing data such that the networks can be tested under different seasonal conditions[1]. Therefore, data from the last 5 days in each month in 2017 are used as testing data. The networks are generally trained on training data according to a limited window $W = \left(M, \frac{M}{2}, f_{update}\right)$ with M days before the desired date in 2017, with $\frac{M}{2}$ days both before and after this date in 2015 and 2016, and with an update frequency f_{update} after which the networks are trained again.

III. Results

For each of the architectures shown in figure 2, several different hyperparameters (e.g. depth, width, number of epochs) can be chosen. This section is divided into 3 phases: in section III-A, a shallow LSTM network is initially tested with different general hyperparameters in terms of the number of (with the default settings in parentheses) AR terms (2), update frequency (14 days), window length (20 days), number of epochs (20), input type (processed), and loss function (MSE), and the hyperparameters giving the lowest overall MAE for the validation data in each test are then chosen and used as a baseline. This baseline is used in the second phase in section III-B, where specific hyperparameters in terms of e.g. activation functions, depth, and width for the architectures shown in figure 2 are tested. The optimal hyperparameters found in sections III-A and III-B are then used on the testing data in the performance analysis in section III-C.

[1]Read more about training, validation, and testing data in the article *How (and why) to create a good validation set* on fast.ai.

When tuning the networks' hyperparameters, the data used to train the networks is as described in section II, and the reserved testing data are excluded.

A. Initial tests

The initial tests are performed in order to find the initial settings for the architectures shown in figure 2. Figure 3 shows the results from tests with varying numbers of autoregressive terms and update frequencies.

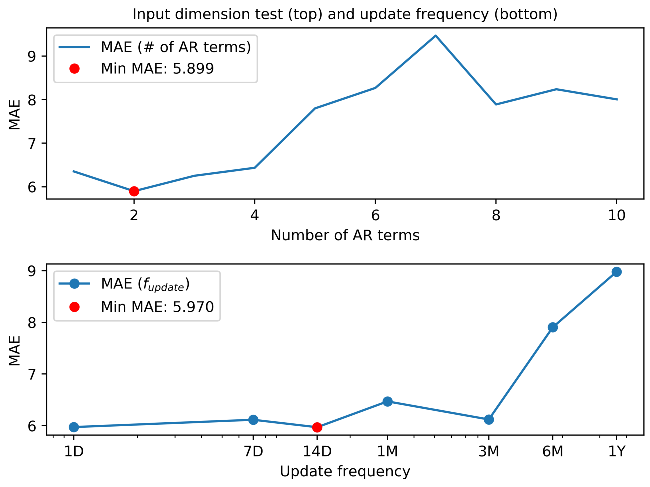

Fig. 3: Test results for input dimension and update frequency. '#' means 'number of'.

The test results in figure 3 show that 2 autoregressive terms is the best choice since the MAE is lowest here. Even though a higher update frequency gives a lower MAE, it is also chosen to train the model every month, which partly is because increasing the training frequency would result in more models to evaluate in the future and partly because the difference is negligible.

Figure 4 shows the test results for varying the window length and number of epochs. It is apparent that both errors stabilise and little to no improvement after 40 epochs and a window length of 70 days can be seen. These values are used in the remainder of this paper.

In section II it was discussed that a zero mean signal would result in a faster convergence, which is seen from table I to be the case when inputting the residual signal. Here the MAE is greatly reduced with the processed input type, whereas the predictions using raw data results in oscillations around the input data with no reasonable results. Table I also shows that minimising loss functions which deals with absolute error measures such as the MSE and MAE greatly outperform relative error measures such as the Mean Absolute Percentage Error (MAPE). The MSE is used as loss function, and the processed data is used as input.

The idea is that these hyperparameters also make the other architectures perform the best since they are all build on the general LSTM architecture but this has not been verified.

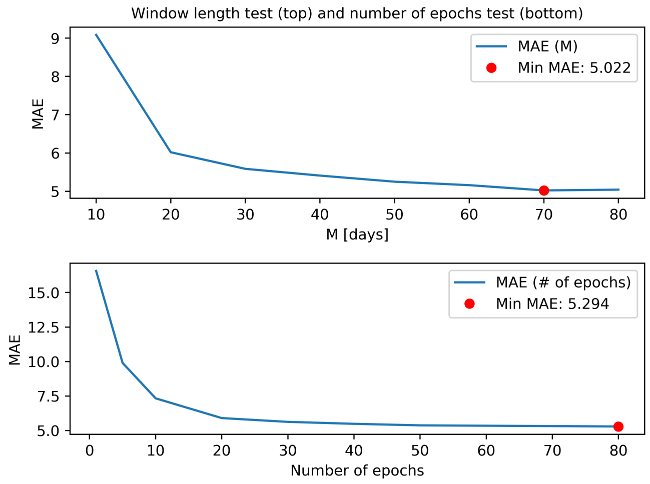

Fig. 4: Test results for the window length and number of epochs.

Input type test			
Input type	Raw	Processed	Raw + processed
MAE	33.948	6.02	24.646
Loss function test			
Loss function	MSE	MAE	MAPE
MAE	5.959	5.961	24.378

TABLE I: Results for the input type and loss function tests. Raw data is the SPP and GHI, and processed data is P_{res} and P_{X}.

B. Architecture tests

The hyperparameters found in section III-A are used by default and are not tested any further.

The test results for different settings of the networks in figure 2 are shown in table II. For each of the networks 3 rows are shown: the first row shows the architecture name and type of hyperparameters, the second row shows the most significant hyperparameters with the default hyperparameter underlined, and the third row shows the MAEs for the hyperparameters with the ones used in the performance analysis in section III-C marked in bold font. Generally, the networks have been tested with other numbers of layers and neurons than the ones presented in this section but the depths and widths shown here give the lowest MAE.

The dLSTM architecture proved worse than the architectures which are just fed with the residual signal, and it is possible that it fails to calculate the residual signal.

The sLSTM architecture can use shortcuts, which means that $\mathbf{x}_t$ is fed directly to each LSTM cell as shown in figure 2. The results from the depth test are somewhat inconclusive because additional layers proved to have little to no benefits. In order to limit the number of weights to be trained the depth is chosen to be 3 layers. When shortcuts are present it allows a more direct flow of information and shows an increase, and thus it will be used in the final sLSTM architecture.

Test results for the general DTDS architecture are split into test results for the DS and DTs architectures. For the DS architecture the state depth was 3 by default (not included in table II). For the DTs network, the widths of 98 and

2 neurons correspond to the 98 municipalities and to the western and eastern part of Denmark, respectively. The input types refer to whether **c**, **h** (as defined in (5) and (7)) or both **c** and **h** are used in the recurrent part of the network.

dLSTM:

Activation function			**Depth**			**Width**		
ReLU	Hard Sigmoid	Tanh	2	3	7	2	17	32
37.78	9.82	**7.82**	**7.03**	7.24	8.56	**7.08**	11.97	13.34

sLSTM:

Activation function			**Depth**					**Shortcut**	
ReLU	Hard Sigmoid	Tanh	2	3	5	7	9	True	False
37.78	10.06	**4.20**	4.24	**4.22**	4.23	4.21	4.29	**4.24**	5.39

DO:

LSTM activation			**FNN activation**			**FNN Depth**		**LSTM output dim**		
ReLU	Tanh	Identity	ReLU	Hard Sigmoid	Tanh	1	2	2	47	98
22.29	13.69	**12.00**	**13.52**	22.85	18.39	**5.30**	11.56	8.58	4.57	**4.39**

DS:

Deep state activation			**State depth**			**Deep state**		**Input types**		
ReLU	Hard Sigmoid	Tanh	1	9	15	True	False	**c**	**h**	**c**, **h**
4.45	5.48	4.44	**4.28**	4.50	4.37	**4.43**	4.45	**4.41**	5.55	5.37

DTs:

Activation function		**Depth**		**Width**		**Deep input**		**Input types**		
ReLU	Tanh	1	3	98	2	True	False	**c**	**h**	**c**, **h**
4.45	20.47	**4.62**	6.28	**4.44**	18.26	**4.43**	4.97	**4.41**	4.45	9.54

TABLE II: MAEs for tests with the architectures shown in figure 2. Different parameterizations have also been tested but the ones shown here are deemed the most significant.

C. Performance analysis

The main interest here is to examine whether an LSTM neural network can outperform the ARX model. Table III shows the performance of the ARX model and 6 different LSTM architectures tested on the test data. The various architectures are fitted with data until the day before the testing data in any given month and with the hyperparameters found in sections III-A and III-B. For the general DTDS architecture the settings are as for DS and DTs in table II except with *Deep state* always set to *True*.

From table III it is immediately seen that the ARX model performs worse than all of the LSTM architectures. The stacked LSTM has the best overall performance but the shallow LSTM has a similar performance. Figure 5 shows selected plots with the overall behaviour of the architectures.

Upon inspection of the plots in figure 5, all models seem to perform reasonably well but some subtle details are seen. For west Denmark in July, the ARX model have noticeable errors before noon. There are similar examples where the LSTM architectures clearly perform worse than the ARX model (e.g. the sLSTM for east Denmark on February 26). Conclusions should not be based on these observations except perhaps that the models have no obvious flaws (such as guessing the mean or some other constant).

IV. Discussion

In this paper different neural network architectures have been implemented and tested with different hyperparameters

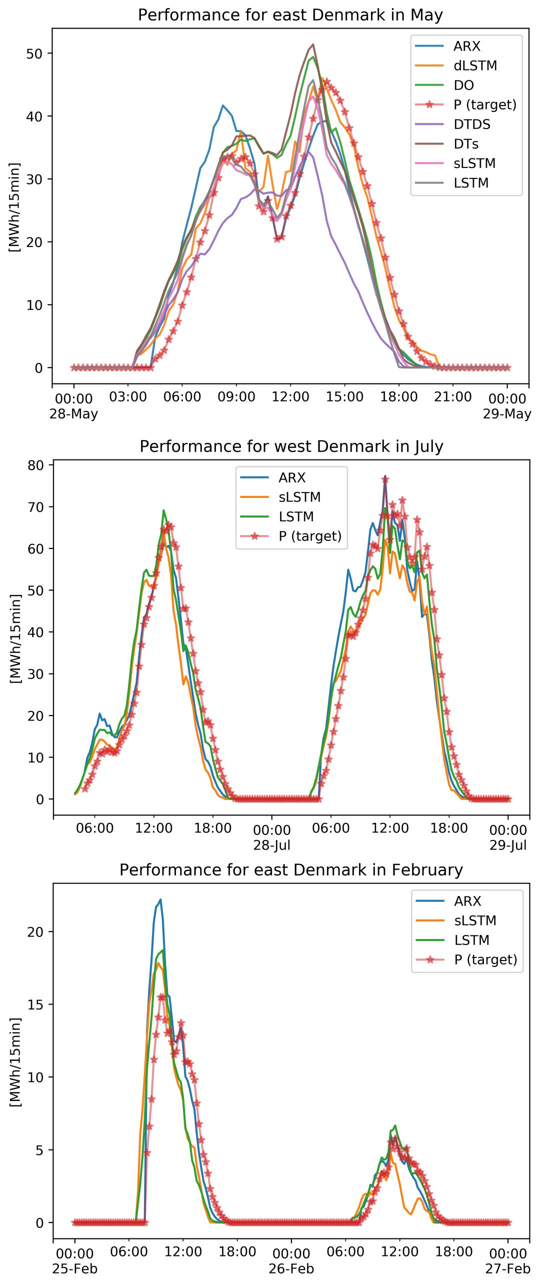

Fig. 5: Examples for predictions on test data in either the eastern or western part of Denmark. Target data are drawn in dark red.

with the purpose of forecasting the solar power production. The MAEs for all hyperparameters in each architecture were averaged across 2017, and a single set of hyperparameters giving the lowest overall MAE were chosen for each architecture. It can however be argued that choosing different hyperparameters for different models (e.g. for each month) could have improved the performance but through analyses of the performance under different hyperparameters in different months it was seen that the improvement would be marginal at best.

Furthermore, the set of hyperparameters constructed in sections III-A and III-B is found by using the hyperparameter that gives the lowest MAE in the corresponding test. This might however result in a local minimum but multidimen-

Model	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Mean	MPD [%]
ARX	6.95	6.65	10.08	14.93	11.02	9.19	10.13	10.85	6.95	7.93	3.72	4.65	8.59	0
LSTM	3.71	3.39	4.51	**5.87**	7.84	7.63	**6.96**	5.04	**5.28**	5.48	2.22	**1.83**	4.98	53.21
dLSTM	4.77	5.13	7.44	9.61	9.04	8.00	8.35	7.92	7.06	8.94	3.18	2.67	6.84	22.68
sLSTM	**2.38**	4.12	**4.32**	7.05	8.37	**5.53**	7.16	**4.73**	5.82	**4.68**	2.09	1.85	**4.84**	**55.85**
DTs	4.44	5.56	4.59	8.78	9.53	6.16	7.20	5.31	7.06	6.12	2.13	2.03	5.74	39.78
DTDS	3.85	**3.31**	4.62	6.45	12.04	6.50	7.21	6.22	6.46	5.07	2.34	1.95	5.50	43.86
DO	4.34	4.00	9.61	6.86	**7.63**	8.29	7.01	5.52	5.85	4.70	**1.61**	2.16	5.63	41.63

TABLE III: Performance analysis for various models. Test data is from the last 5 days in each month of 2017. The values shown are measured as the MAE in MWh/15min. The value marked in bold font in each column shows the model with the best performance. The 'Mean' column represents the average performance on the test data across all of 2017. The last column is the mean percentage difference (MPD) with respect to the ARX model, which is calculated as $\frac{|m_1 - m_2|}{\overline{m}} \cdot 100$, where $\overline{m}$ is the mean difference between the MAEs m_1 and m_2. In some months the target data have outliers, which are excluded from testing data.

sional optimisation has not been performed due to time considerations.

The models' performances are compared in section III but it is generally difficult to conclude whether a model is in fact better than another model (that is, if the improvement is statistically significant).

The performance analysis in section III-C generally showed that the LSTM architectures have better performances than that of the ARX model. The dLSTM network builds on the idea of having an FNN model the exogenous input P_X from the GHI instead of the piecewise linear model currently employed but this type of network did not perform very well. It could however be argued that the input time series are not zero mean, which may result in the system being unable to learn the trends, and further testing involving standardization of the time series is required.

Table III shows that the sLSTM's performance is the best compared to the other models but it is however only a marginal improvement over the shallow LSTM. The DO and DTDS networks have the best performance in two months, which could imply that some part of the time series require a deep transition in order to model the data. But for some of the data the shallow and stacked LSTMs perform the best, which implies that having additional depths might not be a good solution. The trend seen in table III is that the shallow and stacked LSTMs seem most promising. However, which of these two architectures is the better choice remains inconclusive.

V. Conclusion

This paper investigates the short-time forecasting of solar power production and shows that it can be beneficial to use LSTM architectures over a traditional persistent model. Different LSTM architectures have been tested with depth added in different ways, and it is concluded that these architectures result in a general improvement compared to the model currently employed by Energinet. The trend found in this paper is that the shallow and stacked LSTMs are the most promising architectures with improvements of 53.21% and 55.85%, respectively. The difference in performance between these architectures is however marginal, and more testing is required in order to determine the better choice. Based on the results in this paper an approach using the LSTM architecture seems beneficial, and Energinet is advised to look further into this topic.

Other types of networks such as GRU remain untested and might be able to provide similar or better predictions.

Acknowledgements

We would like to thank Energinet for the collaboration and for providing both the solar panel and weather data.